

2

NPSCS-92-010

NAVAL POSTGRADUATE SCHOOL

Monterey, California

AD-A255 896



SD
OCT 05 1992
A D

Object-Oriented Real-Time Computing

Michael L. Nelson, Major, USAF

Aug 1992

92-26368



Approved for public release; distribution is unlimited.

Prepared for:

Naval Postgraduate School
Monterey, California 93943

92 10 2 008

NAVAL POSTGRADUATE SCHOOL
Monterey, California

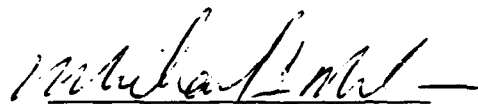
REAR ADMIRAL R. W. WEST, JR.
Superintendent

HARRISON SHULL
Provost

This report was prepared with research funded by the Naval Research Funds provided by the Naval Postgraduate School.

Reproduction of all or part of this report is authorized.

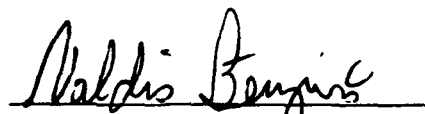
This report was prepared by:



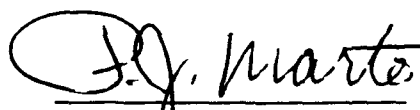
Michael L. Nelson
Assistant Professor of
Computer Science

Reviewed by:

Released by:



VALDIS BERZINS
Associate Chairman for
Technical Research



PAUL MARTO
Dean of Research

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS		
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE			4. PERFORMING ORGANIZATION REPORT NUMBER(S) NPSCS-92-010		
6a. NAME OF PERFORMING ORGANIZATION Computer Science Dept. Naval Postgraduate School			6b. OFFICE SYMBOL (if applicable) CS		5. MONITORING ORGANIZATION REPORT NUMBER(S) Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943			7b. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943		
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Naval Postgraduate School		8b. OFFICE SYMBOL (if applicable) NPS		9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943			10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.
			WORK UNIT ACCESSION NO.		
11. TITLE (Include Security Classification) Object-Oriented Real-Time Computing					
12. PERSONAL AUTHOR(S) Michael L. Nelson					
13a. TYPE OF REPORT Summary		13b. TIME COVERED FROM TO		14. DATE OF REPORT (Year, Month, Day) 1992 August 17	
15. PAGE COUNT 18					
16. SUPPLEMENTARY NOTATION					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP	Object-oriented operating systems, programming, programming languages, real-time computing, real-time systems, simulation		
19. ABSTRACT (Continue on reverse if necessary and identify by block number)					
<p>This paper presents a brief overview of object-oriented programming and real-time systems, followed by an in-depth discussion of object-oriented real-time computing. Examples of object oriented real-time computing systems are included, with special emphasis given to systems developed at the Naval Postgraduate School.</p>					
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT <input checked="" type="checkbox"/> UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS RPT. <input type="checkbox"/> DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED		
22a. NAME OF RESPONSIBLE INDIVIDUAL Michael L. Nelson			22b. TELEPHONE (Include Area Code) (408) 646-2026		22c. OFFICE SYMBOL CSNe

TABLE OF CONTENTS

1	INTRODUCTION	1
2	OBJECT-ORIENTED PROGRAMMING	1
2.1	OBJECT-ORIENTED DESIGN	2
2.2	CONCURRENT AND DISTRIBUTED OBJECT-ORIENTED SYSTEMS	2
3	REAL-TIME SYSTEMS	3
4	OBJECT-ORIENTED REAL-TIME COMPUTING	3
4.1	OBJECT-ORIENTED PROGRAMMING LANGUAGES	3
4.2	OBJECT-ORIENTED OPERATING SYSTEMS	5
4.3	OBJECT-ORIENTED SIMULATION	6
4.3.1	Object-Oriented Simulation of Real-Time Systems	6
4.3.2	Object-Oriented Real-Time Simulation	7
4.3.3	Object-Oriented Simulation Languages	7
4.4	OBJECT-ORIENTED REAL-TIME SYSTEMS	8
5	CONCLUSIONS	9
	APPENDIX: The Actor Model of Concurrent Computation	10
	REFERENCES	11
	INITIAL DISTRIBUTION LIST	14

Accession For	
NTIS	CRA&I
DTIC	TAD
Unannounced	
Justification	
By	
Distribution	
Availability	
Dist	Accession
	Special
A-1	

1 INTRODUCTION

Object-oriented programming (OOP) is a relatively young field that shows great potential in several areas, including real-time (RT) systems. The primary benefits of OOP (encapsulation and reusability) are extremely useful in the RT arena. OOP is also a natural for modeling real-world entities, which increases its attractiveness for RT programming.

This paper presents a brief overview of object-oriented programming and real-time systems.¹ This is followed by an in-depth discussion of object-oriented real-time (OORT) computing, including examples of object-oriented real-time computing here at the Naval Postgraduate School (NPS) and elsewhere.

2 OBJECT-ORIENTED PROGRAMMING

Unfortunately, OOP means many things to many people, creating a rather confusing situation; it has even been said that we have created an "Object-Oriented Tower of Babel" [Nelson91b]. We have found, however, that the following definition encompasses most of the current ideas about OOP [Wegner87, p.169]:

object-oriented = objects + classes + inheritance²

A *class* can be thought of as an abstract data type (ADT). It is used to provide a *template of variables and methods* (operations) for objects. An *object* is then an *instantiation* of a class. *Inheritance* is simply a way of sharing code between classes. A *subclass* inherits all of the variables and methods defined for its *superclass*. Inheritance allows for reusability within and between applications [NF92]. A *class hierarchy* is a set of classes related by inheritance. With single inheritance (usually referred to more simply as inheritance), a class is allowed to have at most one superclass. With *multiple inheritance (MI)*, a class can have any number of superclasses (note that with MI, the class hierarchy is technically a *lattice*, but is still commonly referred to as a hierarchy).

Objects are *encapsulated*. That is, the only way to access an object's variables is via its methods. While this enhances reliability, it often decreases efficiency in manipulating the object, although this is usually not a problem in compiled strongly-typed languages.

¹It is assumed that the reader has some previous knowledge of object-oriented programming and real-time systems. If this is not the case, the interested reader should refer to [Nelson90, Wegner87] for a more in-depth introduction to OOP and to [BBBN92, BW90, SR88] for a more in-depth introduction to RT systems.

²Obviously, this definition excludes delegation-based systems. This definition has been extended as "object-oriented = (objects + classes + inheritance) OR (objects + delegation)" [Nelson90, p.7]. It should be noted that languages may also be classified as object-based "if it supports objects as a language feature" [Wegner87, p. 169]. That is, an object-based language does not necessarily include inheritance. However, it is general practice to refer to a language as object-based if it does not support inheritance and as object-oriented if it does support inheritance.

2.1 OBJECT-ORIENTED DESIGN

Object-oriented design (OOD) often adds to the confusion of OOP as OOD can be used regardless of whether or not the implementation language is object-oriented (OO). Thus we can use an OO approach in designing virtually any system, although it might be argued that implementing an OOD in an OOPL is the 'natural' way to proceed. There are also several schools of thought on how to carry out OOD [Booch91, Bulman92, PN91, WWW90]. However, most approaches to OOD still require an expert from the problem area. [Nelson92]

The Real-Time Object-Oriented Modeling (ROOM) methodology [SGME92] has been specifically developed for designing OORT systems. Three major principles for the development of a RT methodology serve as the basis for ROOM: (1) the key modeling concepts must be domain-specific and intuitive; (2) the methodology must eliminate discontinuities in the development process; and (3) the methodology must support an iterative computer-based development process and an early execution capability. ROOM is based upon a single model that takes into account both the modeling dimensions paradigm (structure, behavior and inheritance) and the abstraction levels paradigm (system, concurrency, and detail). [SGME92]

2.2 CONCURRENT AND DISTRIBUTED OBJECT-ORIENTED SYSTEMS

Although RT systems do not necessarily utilize multiple processors, it is a fairly common way to achieve the required processing power. OOP is a 'natural' for concurrent and distributed systems - virtually any real-world object can be modeled in an OOP system, and as real-world objects can exist and do things concurrently, the objects modeling them can also exist and do things concurrently [Nelson91a, Nelson92]. A "virtual explosion of concurrency" is possible when you consider that several objects may each be executing several methods, and that each of these methods may in turn be executing several operations, with all of this happening concurrently [Nelson91a].

One problem that remains to be solved is how to decide which objects to load on which processors; conventional OOD only determines what your classes/objects should be, not how to distribute them across several processors (discussion at [NATO92]). An optimal solution to the task scheduling problem in a distributed system has been shown to be computationally hard (i.e., NP-complete) [Coffman76, MK84]; as the distribution of classes and objects is a very similar problem, it too is most likely computationally hard.

We have developed an approach that we call the Decomposition Cost Evaluation Model (DCEM). DCEM includes a series of equations that are useful in determining which objects/classes to load on which processors in a distributed system. These equations determine computation and communication costs for objects, classes, and class hierarchies. Although this approach does not guarantee an optimal loading

strategy, it is useful in comparing various alternatives. We then utilize an approach that we call Confined Space Search Decomposition (CSSD) to dynamically adjust load balancing at run time. We have also implemented a distributed dynamic load balancing heuristic that we call Object Reincarnation (OR). Rather than moving objects from one processor to another, an object dies at one site and is then reincarnated at another. [MNK92, Mota92]

3 REAL-TIME SYSTEMS

Real-time is another term that means different things to different people. In general, RT systems have requirements for both logical correctness and timing. Failure to meet these requirements results in some form of 'catastrophic' failure, such as lost or damaged equipment, lost production time, human injury or death, etc. It is important to realize that logically correct results produced too late (i.e., timing requirements not met) may be just as unacceptable as incorrect results produced on time. [BBBN92]

RT systems are often classified as either *hard* or *soft*. Hard RT systems are those systems in which the correctness of the system depends on meeting all of the correctness and timing requirements. Soft RT systems, on the other hand, may still function correctly even if some deadlines are occasionally missed. [BW90]

RT systems often consist of a computer interfaced to some 'critical' system. The computer is typically dedicated to and/or controlling that critical system. The computer in these types of RT systems are often referred to as *embedded systems*, and as such the terms embedded systems and RT systems are often used interchangeably. [BW90]

4 OBJECT-ORIENTED REAL-TIME COMPUTING

4.1 OBJECT-ORIENTED PROGRAMMING LANGUAGES

Unfortunately, most object-oriented programming languages (OOPLs) were not designed for RT applications. That is, much like most conventional languages, they do not contain constructs for enforcing timing constraints. However, in many conventional RT applications, timing constraints are not expressed explicitly in the program, but rather they are described in a separate timing chart [ITM90]; thus non-RT OOPLs are no worse than conventional languages in this regard. All OOPLs do provide the benefits of increased encapsulation and reusability. As previously mentioned, equations to determine computation and communications costs can also be developed, and these could readily be used in developing timing charts. However, the ideal OOPL for a RT application would allow timing constraints to be directly expressed within the program. Such OOPLs will now be discussed.

ACT++ [Kafura88, KL90]. ACT++ is a concurrent OOPL developed specifically for use in concurrent OORT systems. One of the main goals in the design of ACT++ was "to develop a language which supports the powerful actor concurrent computation model and provides software reusability through the class inheritance of an object-oriented language" [KL90, p.25]. ACT++ is thus an implementation of the actor model³ in the C++⁴ OOPL.

Actra [BTDMWL92]. Actra adds the concepts of actors to Smalltalk [PW88]. It "provides an integrated, multi-user, multiprocessor object-oriented programming environment for use in medium and high performance industrial applications" [BTDMWL92, p.1]. Actra actors are active objects which are created and sent messages like ordinary Smalltalk objects, but they can also be activated with independent threads of control. Actors can execute concurrently on a multiprocessor. Actra runs on the Harmony real-time multiprocessor kernel [Barry90, BATW87, MGSW89] which is discussed in the following section.

RTC++ [ITM90]. RTC++ is another extension of C++. Its main features are the ability to specify: (1) a real-time object which is an active entity; (2) timing constraints in an operation as well as in statements; and (3) a periodic task with rigid timing constraints. RTC++ also supports the ability to avoid the priority inversion problem (i.e., having a high priority task wait while a lower priority task executes). RTC++ is designed to run on the ARTS real-time distributed operating system kernel [TM89] which is discussed in the following section.

RTC++ adds two types of objects to C++. An *active object* allows multiple threads (called *member threads*) of control (a conventional C++ object has only a single thread of control). A *real-time object* is an active object defined with timing constraints, which can be specified for an entire operation or for each individual statement. By allowing timing constraints to be expressed within the program, timing errors can be bound at run-time as well as compile-time. When a transaction cannot finish within the specified time, its execution is aborted and alternative instructions are executed in order to satisfy the timing constraints. This is accomplished by an exception handler.

Periodic tasks are specified via a cycle statement. A cycle can be declared with a starting time, ending time, period, and a deadline. Priority inheritance is used to avoid the problem of priority inversion. That is, when a task is providing a service for some client, then the server inherits the priority of that client. Additionally, if a client with an even higher priority is waiting for service, then the server inherits the priority of that waiting client's priority.

³The actor model is briefly discussed in the Appendix.

⁴C++ [Stroustrup86, WP88] is an object-oriented extension of C [KR78].

4.2 OBJECT-ORIENTED OPERATING SYSTEMS

Object-oriented operating systems (OOOS) is another area from which OORT systems can benefit. An OOOS consists of several objects working together to accomplish the tasks normally expected of an operating system. These objects, such as a scheduler, can be replaced at any time by another object which is better suited for a particular application. A real-time operating system (RTOS) is one which is designed specifically for use in RT systems. In this section, we consider several OO RTOSs.

Alpha [Jensen92]. Alpha is an experimental operating system kernel designed for use in distributed RT systems. Scheduling in Alpha is based on the Benefit Accrual Model, in which the overall benefit to the application for completing a certain task is calculated. This approach allows for graceful degradation when all deadlines cannot be met by simply accomplishing the most beneficial tasks. This is also referred to as a "best-effort" scheduling policy.

ARTS [TM89]. ARTS is a distributed OO RTOS intended to provide a predictable, analyzable, and reliable environment. Various scheduling techniques and software tools have been developed for analyzing system schedulability (i.e., for determining whether or not timing requirements can be met). An integrated time-driven scheduling (ITDS) model is used to predict whether or not various tasks can meet their deadlines, and to control which tasks should complete their computations and which should be aborted if all deadlines cannot be met. Priority inheritance, as previously discussed, is used to prevent priority inversion among tasks. Objects can be passive or active. A passive object contains no explicit declaration of a thread which accepts requests. An active object contains at least one such user defined thread (in the case of multiple threads, it is the responsibility of the object's designer to provide concurrency control). Threads can be defined as periodic or aperiodic.

CHAOS [GS89]. A Concurrent Hierarchical Adaptable Object System (CHAOS) is an OO RTOS intended to support the programming of applications that are accountable, efficient, and predictable. Objects can be invoked by either control transfer, data transfer, or a combination of the two. Explicit scheduling parameters can be attached to object invocations. Communication links to objects at different processors can either be specified by the programmer or left up to the system.

Harmony [Barry90, BATW87, MGSW89]. Harmony is designed for use in embedded systems. It is not designed to support program development; this can be done on any host system that has a cross-compiler for the target system. Harmony is written in C, with a 'small amount' of assembly code. Programs can be written in any language capable of linking with the C code in Harmony.

4.3 OBJECT-ORIENTED SIMULATION

OORT systems can also benefit from OO simulation. This can be either: (1) an OO simulation of an RT system, in which an OO approach is used in the simulation of an RT system; or (2) an OORT simulation, in which an OO approach is used in a RT simulation system. OOP is particularly well-suited for simulation systems in that real-world objects and their activities can be simulated as objects with a set of methods to manipulate them. OO simulation also allows for potential changes to a system to be tried out before actual implementation in the real-world system. Both objects and methods can be modified and/or replaced at any time to determine their viability. Hybrid simulation systems are also possible, in which some of the objects in the simulation system are replaced by their actual real-world components.

4.3.1 Object-Oriented Simulation of Real-Time Systems

NPS AUV [BN91, NB92]. The NPS Autonomous Underwater Vehicle (AUV) is an unmanned untethered submarine. Initial software development led to a series of data flow diagrams (DFDs) which modeled both the hardware and software components of the vehicle, along with the flow of data between the components. These DFDs were then used to develop a simulation of the AUV in an OO environment. Although only the simple, high-level DFDs have been included in the simulation to date, it still provides a working model of the AUV which can be used to test control flow and data flow between the various components. The simulation also provides the ability to test modifications and/or replacements of the components at any time.

AMEP [Barry89]. The Advanced Modular ESM Processor (AMEP) is being developed by the Canadian Department of National Defence at the Defence Research Establishment Ottawa. It combines hard RT data acquisition with knowledge-based signal processing and complex user interfaces. Development was based "loosely on the actor model" [Barry89, p.257] and it was implemented in Smalltalk on top of the Harmony OO RTOS. AMEP was built as a prototype system to examine overall viability. Three optimization techniques were used: (1) recoding frequently invoked methods in a lower level language; (2) re-implementing an actor as an independent task in Harmony; and (3) creating "virtual devices" which operate outside of Harmony with an interface much like a hardware device.

LACE [LK92]. Land Air Combat in ERIC (LACE) is an interactive battlefield simulation system (a simplified version of LACE, called Air Combat in Eric (ACE) also exists) developed at Rome Laboratory. Although LACE was developed mainly to test the applicability of the OO approach to battlefield simulations, it does serve three practical purposes: (1) to aid in the air-land battle decision making process; (2) to evaluate Command, Control, Communications and Intelligence related decision aids; and (3) to serve as an air-land battle training aid. LACE was developed in ERIC, an OO simulation language which will be described shortly.

4.3.2 Object-Oriented Real-Time Simulation

NPSNET++ [ZPMW92]. NPSNET is a networked, real-time vehicle simulation system. It is currently comprised of approximately 20,000 lines of C code. We are just beginning to convert this system to an OO system (called NPSNET++), using C++. There are currently over 100 different types of vehicles in the system, and their management has become quite complex. An OO approach will help to encapsulate the parameters of the vehicles. NPSNET is also responsible for maintaining a consistent distributed database representing the 'world,' and an OO approach should allow for the simplification of the network messages necessary to change the world's state (either vehicle state changes or terrain modifications).

4.3.3 Object-Oriented Simulation Languages

Although OOP is particularly well suited for use in simulation systems, most OOPLs were not designed specifically for use in simulation systems. That is, similar to the primary problem of using OOPLs in RT systems as previously discussed, most OOPLs do not contain constructs for enforcing timing constraints. However, a few such languages have been developed, and will be briefly discussed in this section.

AWESIME [Grunwald91]. AWESIME (A Widely Extensible SIMulation Environment) is a library of classes for use in parallel programming and process-oriented simulation applications on computers with a shared address space. It is written in C++ and was developed at the University of Colorado at Boulder.

AWESIME applications typically consist of one or more threads. Threads are instances of classes which are descendants of the class THREAD. The threads are managed by a 'Cpu Multiplexor' (an instance of a descendant of the class CPUMUX) that schedules the various threads on the CPU(s). Three forms of synchronization are provided. Descendants of the class SPINLOCK are used to block a CPU for a thread. Thread-level locking is provided by descendants of the class RESERVEBYEXCEPTION (such as the class SEMAPHORE). Individual resources may also be locked.

Descendants of the class SIMMUX are used to impose a global simulated time order over the threads. Random number generators can be created from descendants of the class RNG. Descendants of the class SAMPLESTATISTICS are available for use in gathering data and computing statistics.

ERIC and DERIC [LK92]. ERIC was developed for use in OO simulation systems at Rome Laboratory. It is designed to support the development of intelligent, discrete event simulations. ERIC is written in the Common Lisp Object System (CLOS [Keene89]). ERIC includes a special object called the Clock, which is used to control the simulation clock. Although simulations developed in ERIC are relatively easy to modify, they do not run in real-time. Therefore, Distributed ERIC (DERIC) is currently under development, also at Rome Laboratory. DERIC is designed to be upward compatible with ERIC. DERIC provides for concurrent execution under control of the Clock object.

4.4 OBJECT-ORIENTED REAL-TIME SYSTEMS

In this section we briefly discuss various RT systems that have been developed using an OO approach.

NPS AUV [BMKMN92, Byrnes93]. We are currently in the process of involving OOP in the NPS AUV itself.⁵ The Rational Behavior Model (RBM) has been proposed as a three-level software architecture for controlling the vehicle. The three levels of RBM are the Strategic, Tactical, and Execution Levels. The Strategic Level is the top-level user interface. It is intended for developing vehicle missions, which are written in some high-level logic language (such as Prolog [Rowe88]). The Execution Level contains the code that interacts directly with the vehicle hardware, and is usually written in C. The Tactical Level is being developed using an object-oriented approach. It serves as the interface between the high-level user interface code of the Strategic Level and the low-level machine interface code of the Execution Level. It is tasked with maintaining all vehicle state information, environmental information, (pre-loaded) mission execution information, and mission log information. Thus, there are four objects residing at the Tactical Level: the AUV model, the world (environment) model, the mission model, and the mission log. At run-time, the Strategic and Execution Levels are only allowed to communicate with the AUV model, which communicates with the world model, mission model, and/or mission log as required. The OO approach provides modular, encapsulated code with a well-defined interface at the Tactical Level.

NPS Radar Data Tracking [Mota92, MNK92]. We have developed an OO approach for radar data processing in a distributed system.⁶ We first developed the Decomposition Cost Evaluation Model (DCEM) to help determine what processor interconnection scheme should be chosen from identified alternatives, and what initial object/class loading scheme should be used. DCEM brings the mapping problem to a higher level of abstraction where the question is which classes should be loaded on which processors rather than which tasks should be loaded on which processors. Communication and computation costs are defined for objects, classes, and class hierarchies. Note that DCEM is not used to identify alternatives, rather it is used to compare previously identified alternatives. We then devised the Confined Space Search Decomposition (CSSD) to perform load balancing at run time. We also included a distributed dynamic load balancing heuristic called Object Reincarnation (OR). Rather than moving objects from one processor to another, they 'die' in one site (reducing its load) and are 'reincarnated' in another site (increasing its load). The OO approach allowed for encapsulated objects with well-defined interfaces, which in turn led to communication and computation cost equations which were useful both in initializing the system and in achieving minimal cost run-time load balancing.

⁵As previously discussed, we have also experimented with OO simulation of the AUV.

⁶Note that this is actually a simulation system as the computer system is fed a stream of simulated radar plots. However, as the radar data tracking system (i.e., the computerized portion of the tracking system) itself could easily be connected to an actual radar unit, it is included here as an OORT system rather than in the previous section on simulation systems.

ARCS [ZJK90, ZS89]. ARCS is an OO approach to RT control systems for remotely operated vehicles (ROVs) and AUVs under development at International Submarine Engineering (ISE), and is sometimes referred to as 'Events and Actions.' Events, actions, and vehicle components are implemented as objects with a well defined set of operations. A list of actions is created that specify what needs to be performed whenever a piece of data is updated. ISE also developed a "fast and small" preemptive scheduler (called the Turbo Scheduler) that is responsible for determining what operation(s) to schedule next. That is, whenever an event (typically an interrupt) occurs, the appropriate actions are scheduled by the Turbo Scheduler. For this reason, the system is also referred to as 'event-based' or 'event-driven.' One of the long-range goals of this approach is to allow the user to configure the system by drawing block diagrams from a library of components. Systems can currently be prototyped by using this library of components approach.

5 CONCLUSIONS

Object-oriented programming is not a 'cure-all' for all of our problems. It is simply a better approach to many (if not all) programming projects, including real-time systems. Inheritance allows for increased system reusability, which is helpful in any application, and encapsulation allows for increased system reliability, which is crucial in real-time systems.

Object-oriented design methodologies, object-oriented programming languages, and object-oriented operating systems are being developed specifically for real-time systems. This enhances the value of utilizing an object-oriented approach in designing and building real-time systems, making such systems even easier to design, build, and maintain.

APPENDIX: The Actor Model of Concurrent Computation

The actor model [Agha86, KL90, Nelson91a, Nelson92, Wegner87] is not object-oriented as it does not support any form of inheritance or delegation; rather, it is more appropriately considered to be object-based. However, it may serve as the foundation for an actor-based OOPL.

An *actor* is simply an object which responds to messages, but it can only respond to a single message at a time. A *message queue* is associated with each actor to hold incoming messages in the order of arrival. An actor has one or more *scripts* which it can use in response to various messages (a script is essentially the equivalent of a method in more conventional OOPLs).

An actor responds to a single method, then 'dies.' One of the things that it must do before dying is to specify a replacement which will handle any additional messages sent to that actor (actually, the replacement responds only to the next message before dying; it too specifies a replacement which will handle the next message, etc). The message queue associated with the original actor is transferred to its replacement. This replacement may be another actor all together, or, more likely, a 'clone' (possibly modified) of the original actor. This replacement may be specified at any time. If the creation of the replacement is specified before the original actor is finished responding to its message, then the replacement may begin to respond to the next message while the original actor is continuing to service the first message.

Concurrency is supported in many ways. Messages can be sent to several actors, so that each is responding to its message. In servicing a message (i.e., carrying out a script), the actor may send messages to other actors. Those actors begin to service their messages immediately, concurrently with one another and with the original actor. Each of these actors may in turn send messages to other actors, and so on. Additionally, any of the actors involved may specify their replacement before completing their message, and this replacement may respond to another message, again sending messages to other actors, creating its own replacement, and so on.

REFERENCES

- [Agha86] G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems*. MIT Press, Cambridge, MA, 1986.
- [Barry89] B.M. Barry. "Prototyping a Real-Time Embedded System in Smalltalk," *OOPSLA'89*, New Orleans, LA, Oct 1989; special issue of *SIGPLAN Notices*, Vol 24, No 10, Oct 1989, pp 255-265.
- [Barry90] B.M. Barry (Workshop Coordinator). "Workshop: Using OOP for Real-Time Programming," *Addendum to the OOPSLA/ECOOP'90 Proceedings*, Ottawa, Canada, Oct 1990; special issue of *SIGPLAN Notices*, Oct 1990, pp 57-66.
- [BATW87] B.M. Barry, J.R. Altoft, D.A. Thomas, and M. Wilson. "Using Objects to Design and Build Radar ESM Systems," *OOPSLA'87*, Orlando, FL, Oct 1987; special issue of *SIGPLAN Notices*, Vol 22, No 12, Dec 1987, pp 192-201.
- [BBBN92] S.M. Badr, R.B. Byrnes, D.P. Brutzman, and M.L. Nelson. *Real-Time Systems*. Naval Postgraduate School, Monterey, CA, Report No NPSCS-92-004, Feb 1992.
- [BMKMN92] R.B. Byrnes, D.L. MacPherson, S.H. Kwak, R.B. McGhee, and M.L. Nelson. "An Experimental Comparison of Hierarchical and Subsumption Software Architectures for Control of an Autonomous Underwater Vehicle," *Symposium on Autonomous Underwater Vehicle Technology (AUV'92)*, Washington, D.C., Jun 1992, pp 135-141.
- [BN91] R.B. Byrnes and M.L. Nelson. "An Object-Oriented Simulation of an Autonomous Underwater Vehicle," *22nd Annual Pittsburgh Conference on Modeling and Simulation, Part 3: Computers, Computer Architecture, Vision, Microprocessors in Education*, Pittsburgh, PA, May 1991, pp 1581-1588.
- [Booch91] G. Booch. *Object-Oriented Design With Applications*. Benjamin/Cummings, Menlo Park, CA, 1991.
- [BTDMWL92] B.M. Barry, D.A. Thomas, J. Duimovich, J. McAffer, M. Wilson, and W.R. LaLonde. "Actra - A Multitasking/Multiprocessing Smalltalk for Industrial Applications," in [NATO92], pp 1-10.
- [Bulman92] D.M. Bulman. "An Objective Survey," *Embedded Systems Programming*, Vol 5, No 3, Mar 1992, pp 20-30.
- [BW90] A. Burns and A. Wellings. *Real-Time Systems and Their Programming Languages*. Addison-Wesley, Reading, MA, 1990.
- [Byrnes93] R.B. Byrnes. *The Rational Behavior Model: A Multi-Paradigm, Tri-Level Software Architecture for the Control of Autonomous Vehicles*. Doctoral Dissertation, Naval Postgraduate School, Monterey, CA, Mar 1993 (draft).
- [Coffman76] E.G. Coffman (Editor). *Computer and Job-Shop Scheduling Theory*. Wiley-Interscience, New York, NY, 1976.
- [Grunwald91] D. Grunwald. *A Users Guide to AWESIME: An Object-Oriented Parallel Programming and Simulation System*. University of Colorado at Boulder, Boulder, CO, Report No CU-CS-552-91, Nov 1991.
- [GS89] P. Gopinath and K. Schwan. "CHAOS: Why One Cannot Have Only An Operating System for Real-Time Applications," *Operating Systems Review*, Vol 23, No 3, Jul 1989, pp. 106-125.

[ITM90] Y. Ishikawa, H. Tokuda, and C.W. Mercer. "Object-Oriented Real-Time Language Design: Constructs for Timing Constraints," *OOPSLA/ECOOP'90 Proceedings*, Ottawa, Canada, Oct 1990; special issue of *SIGPLAN Notices*, Vol 25, No 10, pp 289-298.

[Jensen92] E.D. Jensen. "An Architectural Overview of Alpha: An Object-Oriented Real-Time Distributed OS Kernel," in [NATO92], pp 1-20.

[Kafura88] D. Kafura. "Concurrent Object-Oriented Real-Time Systems Research," *ACM SIGPLAN Workshop on Object-Based Concurrent Programming*, San Diego, CA, Sep 1988; *SIGPLAN Notices*, Vol 24, No 4, Apr 1989, pp 203-205.

[Keene89] S.E. Keene. *Object-Oriented Programming in Common Lisp: A Programmer's Guide to CLOS*. Addison-Wesley, Reading, MA, 1989.

[KL90] D. Kafura and K.H. Lee. "ACT++: Building a Concurrent C++ with Actors," *Journal of Object-Oriented Programming*, Vol 3, No 1, May/Jun 1990, pp 25-37.

[KR78] B.W. Kernighan and E.M. Ritchie. *The C Programming Language*. Prentice Hall, Englewood Cliffs, NJ, 1978.

[LK92] J.H. Lawton and C.D. Krumvieda. "DERIC: A Distributed Object-Oriented Simulation Language," in [NATO92], pp 1-8.

[MGSW89] S.A. MacKay, W.M. Gentleman, D.A. Stewart, and M. Wein. "Harmony as an Object-Oriented Operating System," *ACM SIGPLAN Workshop on Object-Based Concurrent Programming*, San Diego, CA, Sep 1988; *SIGPLAN Notices*, Vol 24, No 4, Apr 1989, pp 209-211.

[MK84] J. Miklosko and V.E. Dotov. *Algorithms, Software and Hardware of Parallel Computers*. VEDA, Publishing House of the Slovak Academy of Sciences, Bratislava, Yugoslavia, 1984.

[MNK92] G.F. Mota, M.L. Nelson, and U.R. Kodres. "Object-Oriented Decomposition for Distributed Systems" (draft).

[Mota92] G.F. Mota. *Radar Data Processing in a Multiprocessor Architecture*. Doctoral Dissertation, Naval Postgraduate School, Monterey, CA, Jun 1992.

[NATO92] NATO Defence Research Group, Panel 11 on Information Processing Technology. *RSG.I Workshop on Object-Oriented Modelling of Distributed Systems*, DREV/CRDV, Quebec, Canada, May 1992.

[NB92] M.L. Nelson and R.B. Byrnes. "A Spiral Model of Object-Oriented Rapid Prototyping," *Technology of Object-Oriented Languages and Systems 8 (TOOLS USA'92)*, Santa Barbara, CA, Jul 1992, pp 111-120.

[Nelson90] M.L. Nelson. *An Introduction to Object-Oriented Programming*. Naval Postgraduate School, Monterey, CA, Report No NPS52-90-024, Apr 1990.

[Nelson91a] M.L. Nelson. "Concurrency and Object-Oriented Programming," *SIGPLAN Notices*, Vol 26, No 10, Oct 1991, pp 63-72.

[Nelson91b] M.L. Nelson. "An Object-Oriented Tower of Babel," *OOPS Messenger*, Vol 2, No 3, Jul 1991, pp 3-11.

[Nelson92] M.L. Nelson. "Concurrent and Distributed Object-Oriented Languages: Promises and Pitfalls," in [NATO92], pp 1-13.

[NF92] M.L. Nelson and K.A. Fontes. "Reusability in Object-Oriented Simulation," *23rd Annual Pittsburgh Conference on Modeling and Simulation*, Pittsburgh, PA, Apr-May 1992 (not yet published).

[PN91] E.G. de Paula and M.L. Nelson. "Designing a Class Hierarchy," *Technology of Object-Oriented Languages and Systems 5 (TOOLS USA'91)*, Santa Barbara, CA, Jul 1991, pp 203-218.

[PW88] L.J. Pinson and R.S. Wiener. *An Introduction to Object-Oriented Programming and Smalltalk*. Addison-Wesley, Reading, MA, 1988.

[Rowe88] N.C. Rowe. *Artificial Intelligence Through Prolog*. Prentice Hall, Englewood Cliffs, NJ, 1988.

[SGME92] B. Selic, G. Gullekson, J. McGee, I. Engelberg. "ROOM: An Object-Oriented Methodology for Developing Real-Time Systems," in [NATO92], pp 1-13.

[SR88] J.A. Stankovic and K. Ramamritham. *Tutorial: Hard Real-Time Systems*. Computer Society Press of the IEEE, Washington, D.C., 1988.

[Stroustrup86] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, Reading, MA, 1986.

[TM89] H. Tokuda and C.W. Mercer. "ARTS: A Distributed Real-Time Kernel," *Operating Systems Review*, Vol 23, No 3, Jul 1989, pp. 29-53.

[Wegner87] P. Wegner. "Dimensions of Object-Based Language Design," *OOPSLA'87*, Oct 1987; special issue of *SIGPLAN Notices*, Vol 22, No 12, Dec 1987, pp 168-182.

[WP88] R.S. Wiener and L.J. Pinson. *An Introduction to Object-Oriented Programming and C++*. Addison-Wesley, Reading, MA, 1988.

[WWW90] R. Wirfs-Brock, B. Wilkerson, and L. Wiener. *Designing Object-Oriented Software*. Addison-Wesley, Reading, MA, 1990.

[ZJK90] X. Zheng, E. Jackson, and M. Kao. "Object-Oriented Software Architecture for Mission-Configurable Robots," *International Advanced Robotics Programme (IARP), 1st Workshop on Mobile Robots for Subsea Environments*, Monterey, CA, Oct 1990, pp 63-73.

[ZPMW92] M.J. Zyda, D.R. Pratt, J.G. Monahan, and K.P. Wilson. "NPSNET: Constructing a 3D Virtual World," *1992 Symposium on Interactive 3D Graphics*, Cambridge, MA, Mar/Apr 1992, pp 147-156.

[ZS89] X. Zheng and S. Srivastava. "Events and Actions: An Object-Oriented Approach to Real-Time Control Systems," *IEEE Pacific RIM Conference on Communications, Computers, and Signal Processing*, Jun 1989.

DISTRIBUTION LIST

Center for Naval Analyses 4401 Ford Avenue Alexandria, VA 22302-0268	1 copy
Defense Technical Information Center Cameron Station Alexandria, VA 22314	2 copies
Director of Research Administration, Code 81 Naval Postgraduate School Monterey, CA 93943	1 copy
Library, Code 0142 Naval Postgraduate School Monterey, CA 93943	2 copies
Dr. G. Bradley Operations Research Dept., Code ORBz Naval Postgraduate School Monterey, CA 93943	1 copy
LCDR D.P. Brutzman, USN Operations Research Dept., Code ORBr Naval Postgraduate School Monterey, CA 93943	1 copy
MAJ R.B. Byrnes, USA Dept. of Computer Science, Code CS Naval Postgraduate School Monterey, CA 93943	1 copy
Dr. R.B. McGhee Dept. of Computer Science, Code CSMz Naval Postgraduate School Monterey, CA 93943	1 copy
Dr. M.L. Nelson, MAJ, USAF Dept. of Computer Science, Code CSNe Naval Postgraduate School Monterey, CA 93943	40 copies